Trabalho de Implementação 2 - Heurísticas e Metaheurísticas

Francisco Teixeira Rocha Aragão - 2021031726

Data: Dezembro de 2024

1 Introdução

O presente trabalho busca resolver de maneira aproximada o problema do caixeiro viajante (TSP), fazendo uso da metaheurística VND como implementação. Como o problema pertence a classe NP, faz-se necessário o uso de tais estratégias, sendo utilizado no trabalho heurísticas construtivas para definição do caminho inicial, além de diferentes funções de vizinhança para implementar o VND. Abaixo encontra-se mais informações sobre a implementação além dos resultados obtidos.

2 Heurísticas utilizadas

Primeiramente sobre a heurística utilizada, a estratégia implementada para definição inicial de uma solução refere-se a uma heurística construtiva, ou seja, uma heurística em que a solução é construída do zero, desde o início até a resolução do problema. Iniciando-se assim de uma solução vazia, obtendo então uma solução parcial a cada iteração em que ao final é transformada em uma solução completa válida.

Desse modo, a estratégia utilizada foi baseada em uma abordagem gulosa, em que a cada ponto (ou cidade), o próximo trajeto escolhido é aquele com a menor distância. O início é feito a partir de uma cidade (será explicado mais frente) e a cada iteração novas cidades são adicionadas no caminho até todas as cidades serem visitadas, voltando assim ao vértice inicial resolvendo o problema. Com isso, garante-se a validade da solução retornada, em que a cada iteração acrescenta-se uma nova cidade não visitada anteriormente, terminando o algoritmo até visitar a última cidade, retornando ao ponto inicial. Sobre o ponto inicial escolhido, pode ser escolhido tanto iniciar da primeira cidade quanto da cidade central, sendo a segunda abordagem escolhida por apresentar melhores resultados na prática.

Após isso, foi implementada a metaheurística VND, cujo obtivo é, após encontrar uma primeira solução, utilizar diferentes funções de vizinhanças para melhorar a solução obtida, fato motivado pelo problema de ficar presos em ótimos locais, com as diferentes vizinhanças ajudando a diversificar as soluções. Para a implementação do VND, foram utilizado 3 funções de vizinhança diferentes, sendo elas a troca de 2 cidades, a troca de 3 cidades e a troca de 4 cidades. A troca de 2 cidades (2-opt) funciona trocando apenas dois caminhos da solução inicial, enquanto a troca de 3 cidades (3-opt) troca 3 caminhos. A troca de 4 cidades (4-opt) foi implementada seguindo a estratégia de Double Bridge, em que 4 caminhos são trocados, porém são ligados em forma cruzada.

Assim, o VND atua a partir da solução inicial da heurística construtiva e então é aplicado cada uma das diferentes funções de vizinhança para tentar melhorar a solução obtida (de modo a evitar possíveis mínimos locais). Como o número de vizinho em cada vizinhança aumenta exponencialmente (2-opt possui $O(n\hat{2})$ vizinhos, 3-opt possui $O(n\hat{3})$ vizinhos e 4-opt possui $O(n\hat{4})$ vizinhos), a implementação do VND contêm duas estratégias para otimizar sua execução. Primeiramente, é adotado a abordagem primeiro-aprimorante, em que ao encontrar um vizinho melhor, automaticamente finaliza-se a busca nessa vizinhança, reiniciando então o processo. Isso é feito para evitar ficar muito tempo preso em cada vizinhança (custoso para 3-opt e 4-opt). Além disso, ao encontrar a solução, é feito um reinício, voltando a execução da vizinhança 2-opt, de modo, novamente, em passar mais tempo em uma vizinhança mais rápida na esperança de encontrar novas soluções novamente. Com isso, caso a solução passar pelas 3 vizinhanças e não encontrar uma solução melhor, o algoritmo é encerrado, retornando o caminho encontrado.

3 Execução e Resultados

O código foi desenvolvido em C++ e os testes foram realizados em uma máquina com debian 12, 16GB de ram e processador I5-11 geração. Sua execução pode ser realizada com os seguintes comandos:

```
// compilação
make

// limpar arquivos gerados
make clean

// rodar programa
make run ARGS="<pasta com instâncias de entrada> <tipo da cidade inicial>
// <tipo de cidade inicial> = 0 para usar a primeira cidade e 1 para usar cidade central
```

Vale destacar que os arquivos de entrada foram encontrados no site TSPLIB95, presente nas referências no trabalho, com o projeto executando apenas as instâncias que terminam com a extensão '.tsp'. Além disso, a execução foi realizada 5 vezes para cada instância, com as médias dos resultados disponível nas tabelas abaixo. As heurísticas implementadas no trabalho são deterministícas, não obtendo mudanças entre as execuções, então a média dos resultados foi obtida para encontrar o valor médio do tempo de execução.

Desse modo, observando a tabela 1 presente abaixo, é possível visualizar os resultados do VND para as instâncias de teste utilzadas. Assim, percebe-se que para todos os casos, a aplicação do VND conseguiu melhorar o custo final do caminho encontrado, concluindo assim o objetivo de evitar resultados presos em ótimos locais. Embora o custo seja mais alto, em instâncias pequenas o tempo não foi um fator tão relevante, possuindo um maior custo visível nas maiores instâncias (200 cidades).

Instância	Ótimo	Caminho Inicial	Custo Final	Tempo (s)	Erro Percentual (%)
pr144	58537	64161.5	62158.8	40.8551	6.19
pr152	73682	80161.3	78562.3	25.7106	6.61
pr124	59030	68321.4	63340.8	9.61927	7.30
st70	675	783.72	741.092	0.664765	9.79
pr107	44303	52145.5	48863	6.19966	10.30
pr76	108159	151142	119562	1.72431	10.56
kroE100	22068	26884.6	24630.8	7.48277	11.61
pr136	96772	116045	109462	20.6047	13.10
kroC100	20749	24173.9	23554.8	3.51075	13.53
rat99	1211	1493.92	1380.22	10.5665	13.98
kroB100	22141	27208.5	25329.5	7.64223	14.40
kroA100	21282	25781.6	24389.5	4.09814	14.62
berlin52	7542	9140.13	8764.12	0.338102	16.20
kroA150	26524	32786.8	31135.4	45.7727	17.38
kroB150	26130	34583.7	30711.4	48.4881	17.51
kroB200	29437	36893.6	35374	117.834	20.17
rat195	2323	2830.1	2791.6	142.556	20.19
kroA200	29368	37701.5	35637.6	124.522	21.31
lin105	14379	19182.4	18084.7	8.57969	25.78
kroD100	21294	27695.5	27127.5	6.90666	27.41

Table 1: Resultados VND - Custo Inicial, Final e Erro Percentual

Na tabela 2 foram feitos testes em instâncias maiores do problema TSP. É possível observar o aumento no tempo, causado pela quantidade de verificações feitas pelas diferentes vizinhanças que passa a se tornar um grande problema. Vale destacar a diferença de tempo observada entre a tabela 2 e os resultados da tabela 3, com a solução inicial (utilizando uma heurística gulosa) é obtida bem rápido, enquanto o VND com as verificações em multíplas vizinhanças consomem quase a totalidade do tempo de execução.

Instância	Ótimo	Caminho Inicial	Custo Final	Tempo (s)	Erro Percentual (%)
pr299	48191	60288.6	58182.8	726.937	20.75
lin318	42029	53456.1	51718.9	2744.43	23.07

Table 2: Resultados VND - Instâncias maiores - Custo Inicial, Final, Tempo e Erro Percentual

Instância	Primeira Solução (s)
pr299	0.067662
lin318	0.078952

Table 3: Tempos da Primeira Solução por Instância

4 Conclusão

Observando os resultados presentes na tabela acima, percebe-se como o VND é útil e importante em problems com um grande espaço de busca, conseguindo obter ganhos no resultado obtido a partir da troca de vizinhança utilizada. Dessa forma, é possível encontrar melhores soluções a cada nova vizinhança utilizada, ao custo de maior tempo de execução verificando cada vizinho das vizinhanças maiores. Com isso, possíveis trabalhos futuros podem implementar melhorias no código de forma a otimizar o tempo de cada busca nas vizinhanças, ou então testar diferentes vizinhanças para buscar melhores resultados.

5 Referências

VND e Vizinhanças TSP
4-opt e Double Bridge
Artigo: Vizinhanças e busca local para TSP Site com instâncias e ótimos de TSP